# A New Paradigm for Parameterization in Atmospheric Models: Application to the New Fu-Liou Radiation Code

Toshihisa Matsui, Giovanni Leoncini,
Roger A. Pielke Sr., and
Udaysankar S. Nair

Supported by NASA CEAS Fellowship No. NAG5-12105,
NASA Grant No. NNG04GB87G, and
DOD Cooperative Agreement DAAD19-02-2-0005

Colorado State University

DEPARTMENT OF ATMOSPHERIC SCIENCE

PAPER NO. 747

# A New Paradigm for Parameterization in Atmospheric Models: Application to the New Fu-Liou Radiation Code

by

Toshihisa Matsui, Giovanni Leoncini, and Roger A. Pielke Sr.
Department of Atmospheric Science
Colorado State University
Fort Collins, CO 80523

Udaysankar S. Nair
National Space Science and Technology Center
University of Alabama at Huntsville
320 Sparkman Dr., Huntsville, AL 35805

Supported by NASA CEAS Fellowship No. NAG5-12105
NASA Grant No. NNG04GB87G, and
DOD Cooperative Agreement DAAD19-02-2-0005

Atmospheric Science Paper No. 747

July 26, 2004

**Abstract.** This report presents a new paradigm for atmospheric parameterization through the preparation of universal look-up-tables (LUTs) prior to their application in model runs. Instead of repetitive computation of a parameterization during model integration a LUT-based approach selects the response of the parameterization which would have been obtained if the original parameterization were applied. The New Fu-Liou radiation parameterization is presented to illustrate this technique. When the LUT-based approach is implemented, the New Fu-Liou radiation code becomes 443 times faster than the original code. Although the LUT approach require the large disk space, this simple, powerful methodology should be widely applied for various atmospheric parameterizations in the weather forecasting and climate process simulations.

## 1. Introduction

Numerical weather forecasting models predict the evolution of the atmosphere from its initial state by numerical integration of physical equations (*dynamic core*). Simplified mathematical frameworks (*parameterizations*) are then embedded within the dynamical core to represent atmospheric processes that cannot be simulated explicitly and/or accurately due to their high computational costs. This includes the subgrid phenomena of cumulus convection, cloud and precipitation microphysics, radiative transfer, subgrid turbulence, land-surface hydrology and vegetation processes. As generally applied, these parameterizations are vertical column or one-dimensional models (e.g., for microphysics of clouds). During model integration, these 1-D parameterizations are computed repeatedly; e.g., a two-day numerical weather forecast with 100×80 horizontal grid points, one-minute time-steps, and 50 ensemble members requires 576,000,000 repetitive computations of a single parameterization!

The computational burden associated with parameterizations, however, is becoming a more significant issue for many reasons: parameterization complexity is increasing in order to improve accuracy of unresolved processes; the number of parameterizations in a single NWP model is also increasing to represent more processes;

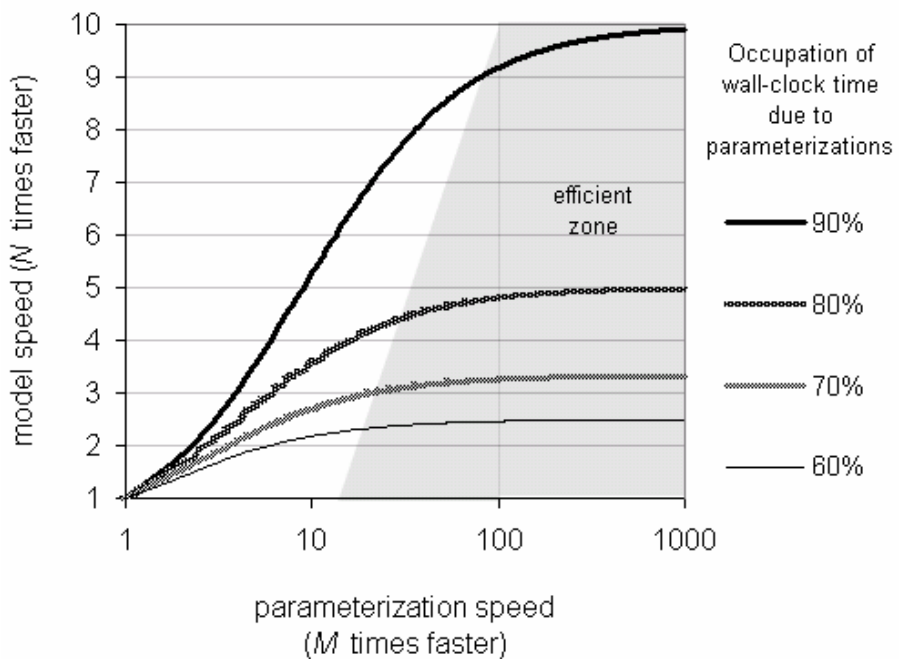


Figure 1. Relationship between improvement of parameterization speed and model speed.

and operational weather forecasting is utilizing finer grid spaces and a larger number of ensemble simulations.

In recent atmospheric process models, parameterizations occupy a large fraction of wall-clock time in a simulation, up to 90%, for example. Simply, as speed of parameterizations are improved, model speed quickly improves. Figure 1 shows a relationship between parameterization efficiency and model efficiency for different wall-clock times due to the parameterization. This figure shows that when a

parameterization become *M* times faster than the original version (*parameterization speed*), the model becomes *N* times faster (*model speed*) than the original model. The *efficient zone* is defined as being 90% of the maximum efficient state (100%; i.e., the wall-clock time of the parameterization is negligible compared with that of the dynamic core). Figure 1 also shows that a heavily-parameterized model needs a greater improvement of parameterization speed.

The computational burden of parameterizations can be reduced in two different ways: 1) reduce the level of complexity of the parameterizations as used within model integration; or 2) pre-compute every possible output of the parameterization in order to transform the parameterizations into massive look-up tables (LUTs) [*Pielke* 1984].

The first methodology has been implemented in the early stages of parameterization development [e.g., *Stephens* 1984], but such a simplification results in a loss of accuracy. The second methodology has been implemented in subsections of many parameterizations [e.g., *Walko et al.* 1995], yet it has not been implemented for entire parameterizations.

Existing techniques have only a modest benefit with increased efficiency (such as some simplification or LUT approach in a subsection of a parameterization) which could

improve the speed of parameterization up to a factor of ~5. In order to maximize the efficiency of atmospheric models (to be in the efficient zone), however, we need to speed up all computationally demanding parameterizations used in the model by a factor of more than 20~100 (Figure 1). Such an improvement would make the wall-clock time of parameterizations negligible compared with that of the dynamic core: e.g., an ideally efficient model. But how can we achieve such an improvement?

This report presents a new platform for parameterization based on the second approach; a "universal" LUT, where the word *universal* indicates that the LUT must span the entire range of every input variable. The method we propose uses a UNIX-based database management type of concept, in order to permit the rapid acquisition of the output variables as a function of the input variables [*Rosen et al.* 1996]. Such an approach is routine in business, but has not been applied in atmospheric modeling. The combination of developing the LUTs, and the quick access to needed values, provides the opportunity for a quantum improvement in the calculation speed of the models. This new platform is particularly useful for a simulation that does not require a modification of parameterizations and frequent changes in the vertical grid structure. Operational weather forecasting and long-term climate simulations represent types of modeling where the LUT

approach would substantially increase computation speed.

The New Fu-Liou (NFL) radiation code [*Rose and Charlock* 2002] is applied for this exercise. Throughout the discussion, the Sun-Blade-1000 (Dual UltraSPARC-III+) (Dual CPU: 900 MHz frequency and 8 Mbytes cash size, Memory size: 4GB, HD: 60 Gbytes) is used to count wall-clock time of computation. The following sections describe the step-by-step methodologies that develop the LUT-based approach using a simple example to illustrate the methodology. We also describe the actual Bourn-Shell and FORTRAN code in the Appendix, although the described methodology is more general and independent of hardware and software.

## 2. Method of LUT-based Approach to Parameterizations

### a. Input and Output Variables

A parameterization has numerous input variables, and "tunable" or experimentally determined coefficients that are used during the computation, but are not fed back into the atmospheric model. The former are considered input to the parameterization, while the latter part of it. We can then, symbolically write:

$$(B_1, B_2, B_3, \ldots B_N) = f(A_1, A_2, A_3, \ldots A_M) \qquad (1)$$

where $B_i$ are output variables, $A_i$ input variables and $f$ represents the parameterization. The number of input and output variables depends on the parameterization, on the numbers of atmospheric model vertical levels, and on the interconnection with other components of the atmospheric model.

As a result the original parameterization must be evaluated offline, for all conceivable combinations of the input variables in order to obtain every possibly useful output, which is then stored and read when needed without any further computation. The number of combinations depends on the number of bins for each input, which is determined by i) sensitivity to the output in the case of continuous input (e.g., real variables), and ii) the possible combination of input variables in the case of the discrete input (e.g., integer variables). *Note that the accuracy of any parameterization need not be any more precise than the least accurate parameterization of significant atmospheric processes of interest* [*Pielke* 2001]. Thereby, to have maximum efficiency without losing accuracy, the range of the input variables and bin size must be found by examining the sensitivity of the output variable to the input variables. A second constrain on the total size is given by the storage capacity of the computer that will actually use the LUT. From our experience 100 Gbytes can be handled by most high performance workstations. Therefore

if each binary file that constitutes one parameterization output occupies 100 bytes, the number of the universal cases must be

less than one billion so as to store on the hard disk. Section 3 describes several solutions for these issues.

**b. Developing Input-Oriented Hierarchical Directories and Files**

The number of the bins and their combinations is usually too large to store the universal LUTs in current computer memory (otherwise, see Appendix 3). Therefore, we develop input-oriented hierarchical directories in order to organize the universal LUTs. This data organization ensures the most efficient process to search the LUT with a set of given input variables (see the next section). The following three processes are simultaneously executed through an input-oriented giant loop of a parameterization: i) convert a set of input variables into a directory name, ii) generate the assigned directory, and iii) compute a parameterization with a set of input variables to store the binary LUT values, which contain a set of output variables, at the end of the assigned directory. For example, the hierarchical directories for a parameterization are

$/ A_1 / A_2 / .../ A_N / LUT.dat$ ,

where *LUT.dat* is binary data that contains a set of output variables ( $B_1, B_2, B_3, ..., B_M$ ). The directory names consist of a combination of digits, e.g., /12/7/92/…./83/. So, in this case the first directory corresponds to the 12th bin of the first input variable, and so forth.

We convert all the input variables into the same number of directory layers as the number of input variables. The above example has *N* layers of directories. For a 5000-time repetitive process (this assumes one time step for the atmospheric model with the number of horizontal grid, 50×100), the wall-clock time of a FORTRAN's input process increases $3.548\times10^{-3}$ (seconds) per one additional directory layer (see the next section). Although this is not significant, it is possible to reduce the depth of the directories through the merging of multiple names for a directory or file: e.g.,

$/ A_1 A_2 ... A_i / A_{i+1} ... A_N .dat$ ,

where $A_{i+1.....} A_N .dat$ is the name of a binary file that contains a set of output variables ( $B_1, B_2, B_3, ..., B_M$ ). In this case, we only have one layer of directories ( $/ A_1 A_2 ... A_i /$ ) and a large number of output files ( $A_{i+1.....} A_N .dat$ ) in the assigned directory. Again, the directory and file names consist of a combination of digits: e.g., / 12792….83 / 3847593287.dat. The choice between the different way of building directory and file names depends on the user and on the limitations of the file system in use. For example, for some versions of the

UNIX System, a file name has a length limitation, or two filenames are considered the same if they agree in the first 14 characters.

Figure 2 shows the schematics of the hierarchical directory. Appendix 1 shows Bourn-Shell and FORTRAN code to execute the combined processes.

**c. LUT-based Approach to Parameterization**

Once the LUT files are stored at the end of the hierarchical directories, we implement the LUT-based approach. The key of the LUT-based approach is to search the correct LUT from billions of the universal LUT values. The LUT-based approach is quite simple and has only two processes; 1) convert a set of the input variables (real or integer variables) into the directory and file name (character variable); and 2) open-read-close the binary LUT in the hierarchical directory. These two processes would be the most efficient way to instantaneously search the LUT from the universal LUT values, and this is why we develop the input-oriented hierarchical directories to organize the huge database. Appendix 2 shows the FORTRAN code to execute the LUT-based approach.

With respect to the FORTRAN input process speed, the size of the binary LUT values should be a minimum. The wall-clock time of FORTRAN's open-read-close

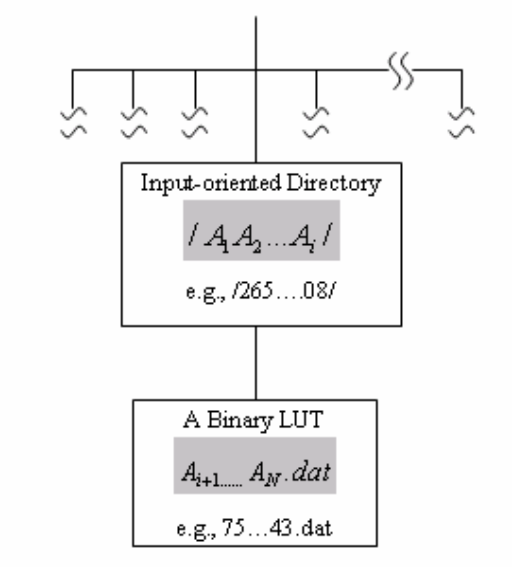


Figure 2. Schematic of hierarchical directories given by a set of input variables, $A_n$ ($n=1,2,\ldots,N$).

processes linearly increases with the size of a LUT value, and its relationship is expressed by

$$Time(\text{sec}) = 0.0001 \times [file\ size(byte)] + 0.1651$$

in the workstation used in this study (the Sun-Blade-1000). If ten output variables are stored in a single binary file, the size of each LUT value becomes 40 bytes. With this LUT, the 5000 repetitive processes of the FORTRAN opening/reading/closing the LUT file takes just 0.1691 (seconds); this is the basic wall-clock time unit of the LUT-based approach. Of course, we should account for other processes that convert the variables into the directory and file name and the depth of hierarchical directories. Thus, the smallest size of LUTs should be stored at the end of the hierarchical directories. If the FORTRAN

open-read-close process for a single LUT value (minimum size) is not significantly faster than the original code, the LUT-based approach will not significantly improve computational speed. This can happen for a parameterization that has a small number of input variables and combinations of all conceivable cases. In this case, data could be stored in FORTRAN code in order to further accelerate the delivery of the results (see Appendix 3).

**d. Application to the New Fu-Liou Code**

The New Fu-Liou (NFL) radiative code was originally developed as a remote sensing algorithm by the Clouds and the Earth's Radiant Energy System (CERES) team at NASA Langley [*Rose and Charlock* 2002]. Whereas the NFL code more accurately accounts for the radiative effect of various hydrometeor through the delta-four stream correlated *K*-distribution scheme [*Fu and Liou* 1992, 1993], it is computationally demanding for numerical weather forecasting due to its complicated structure. The characterization of aerosols requires a combination of ground and *in situ* measurements [*Wang et al.* 2004], and the performance of the NFL model and aerosol characterization has been validated against ground measurements [*Wang et al.* 2004]. We introduce the LUT-based approach to the NFL parameterization in order to speed up the original NFL

code.

We simulated the NFL code with 30 vertical levels, which involve approximately 140 input variables, and for each of the 30-vertical levels, the temperature tendencies and 3 other outputs are required to interconnect the NFL code to other parameterizations and to the dynamic core in the Regional Atmospheric Modeling System (RAMS) [*Pielke et al.* 1992]. If we store 33 output variables in the binary LUT value, 5000 repetitive processes of the opening/reading/closing of the LUT file (132 bytes) take just 0.25 (seconds). Indeed, 5000 repetitive processes of i) the statement to convert the 140 inputs into the directory name and ii) open/read/close statement with multi-name 10 layers of directory depth require the additional wall-clock time of 0.05 and 0.035 (seconds), respectively. Therefore, the total wall-clock time of the LUT-based approach to the NFL will be 0.335 (seconds) for 5000 repetitive processes, whereas 5000 repetitive processes of the original NFL code take 148.63 (seconds). *Thereby, as evaluated in this example, the LUT-based* approach to the *NFL is 443 times faster than the original NFL code.*

**e. Implementation Techniques with Limited Disk Storage**

This section describes the feasible solution for limited disk space. As we discussed, NFL code contains about 140 input variables to compute the atmospheric

heating profile. The number of all conceivable cases (including the unphysical cases) would be the too large to store in the database on the typical existing hard drive. Similarly, as the size of bins decrease, the number of LUTs increases. Thus, we propose three possible contingency methods that enable LUTs to fit on the hard drive.

i) *Hybrid Parameterization*: This scheme permits a simulation to choose either the original parameterization code or the LUTs, depending on the availability of LUTs. For example, clear skies, in addition to cloudy skies with warm clouds and/or cirrus clouds generally are most frequently found in the general atmospheric environment. These combinations are generally characterized with less variations and number of input variables. Alternatively, if we account for a rare atmospheric situation, such as deep cumulus convection in the radiation LUTs, we need to include a multi-layer cloud structure and a number of hydrometeor inputs, which significantly increases the size of total LUTs. Instead of generating LUTs for such rare situations, we let the simulation use the original parameterization code if the corresponding LUT is not available. We must investigate how frequent such cases are using the atmospheric sounding datasets.

ii) *Regional/Seasonal Subsetting*: If model runs are limited to specific areas and/or

periods of the year, the range of input variables can be limited to those which are expected to occur. We can develop an infrastructure to develop a main LUT bank, which can be segmented and downloaded for particular seasons and regions via ftp.

iii) *Polynomial Fitting*: The LUT approach, as it has been described thus far, requires the storage of the parameterization output as a function of the input variables, but a polynomial fitting can be used to interpolate it, reducing further the necessary storage space. One issue to consider is whether the smoothing character of the polynomial smears important physical responses (such as associated with a cloud top or bottom), although the issues could be solved by a combination of several polynomial equations.

iv) *Operational Optimization*: During the operational forecasting, a model would check if a LUT is present for a given set of input variables. If the corresponding LUT is not available in the hard disk, a model chooses the original parameterization (*hybrid Parameterization*), then output and a set of input variables are stored in the hierarchical directly. Simultaneously, frequency of access to the LUT will be monitored with the shell script, and facilitate the optimization. Thereby, a model operationally optimizes its database.

Effectiveness of these methodologies varies with different modeling platforms and parameterizations, so they must be developed for each specific configuration.

**3. Summary**

This report describes the development and performance of a new LUT-based approach to parameterizations. The introduction of hierarchical directories and the rapid conversion of input variables into the directory and file names result in a dramatic improvement of computational time compared with the original parameterization. This provides an opportunity for a quantum improvement in the calculation speed of the parameterization and models.

With the LUT approach, the Fu-Liou parameterization possibly becomes the fastest radiation parameterization in RAMS even compared with the simpler radiative code (*Mahrer and Pielke* 1977), while it still contains a more accurate treatment for the radiative effects of hydrometeors and aerosols through the delta-four stream correlated *K*-distribution scheme.

The application of the LUT approach, in lieu of current parameterization methods, is clearly beneficial for operational types of atmospheric modeling. The LUT method removes the need for the countless repetitive computations of a parameterization. In other

words, if we develop the LUT through using the most accurate model (e.g., line-by-line radiation model, spectra-bin Cloud Resolving Model - CRM ) or observations (in-situ or remote sensing), it will be *the most accurate and fastest* parameterization available. There is a need, therefore, to construct such databases for all parameterizations.

Existing computers are already capable of implementing the LUT method for all parameterizations. The only major shortcoming is the size of the universal LUT value. While the LUT-based approach minimizes wall-clock time of the numerical simulation, it requires huge space on a hard disk. We proposed three methodologies to reduce the required storage space: *Hybrid Parameterization*, *Regional Subseting, Polynomial Fitting, and Operational Optimization.* These methodologies could enable existing workstations to apply the LUT approach to any parameterization. Indeed, due to its intrinsic structure, this parameterization is *not tunable*. Thus, only well-validated/calibrated parameterizations should be used for this approach, and operational numerical simulations will be most suited for this approach.

In the near future, we will convert all the atmospheric parameterizations used in RAMS into the LUT-based approach. We will collaborate with various organizations to construct the most reliable universal LUTs, and make it a public-accessible database.

So-called LUT-based Atmospheric Modeling Systems (LAMS) allow one to run atmospheric simulations with more ensemble members with finer grid-incremental spacing in a more accurate manner. Together with the massive parallelism of computation, our least CPU-demanding LUT-based approach could enable an existing supercomputer to implement global cloud-resolving weather forecasting without waiting a few decades.

**Acknowledgements**. This work is funded by NASA CEAS Fellowship No. NAG5-12105, NASA Grant No. NNG04GB87G, and DOD Cooperative Agreement DAAD19-02-2-0005. This report resulted from a course project in AT730 Mesoscale Modeling instructed by Professor Roger A. Pielke Sr**.** The authors thank Robert Walko, Craig Tremback, William Cotton, and Roni Avissar for useful discussions. The authors also thank Dallas Staley for her professional editing.

**Appendix 1**

The following B-Shell and FORTRAN code is used to develop the hierarchical directories to store the binary LUTs. You must insert specific values on @ according to the bin of input variables.

```
#####################################################################
#!/bin/sh
# Initialize the input variables
```

$A_1$ = @; $A_2$ =@ ; …; $A_i$ =@; $A_{i+1}$ =@; ...; $A_N$ =@

# Define the name of executable of parameterization. a.out is executable of

# FORTRAN code (described below).

exec=a.out

# Start the Giant Loop

while [ $ $A_1$ -le @ ] ; do

( while [ $ $A_2$ -le @ ] ; do

…

( while [ $ $A_i$ -le @ ] ; do

#Assign the directory name as DIR

DIR="$ $A_1$ $ $A_2$ …$ $A_i$ "

# create input-oriented directories

[ -r $DIR ] || mkdir $DIR

(while [ $ $A_{i+1}$ -le @ ] ; do

…

( while [ $ $A_N$ -le @ ] ; do

# Store the input variables in an “input” file.

[ -r input ] && rm input

echo $ $A_1$ >> input ; echo $ $A_2$ >> input ; …

echo $ $A_i$ >> input; echo $ $A_{i+1}$ >> input ; …; echo $ $A_N$ >> input

# Run the 1-D parameterization with a set of given input variables

$exec < input

# move the binary LUT at the end of the directly

mv *.dat $DIR

# End of the looping, and updating the input variables.

$A_N$ =`expr $ $A_N$ + 1` ; done ) #loop of $A_N$

…

$A_{i+1}$ =`expr $ $A_{i+1}$ + 1` ; done ) #loop of $A_{i+1}$

$A_i$ =`expr $ $A_i$ + 1` ; done ) #loop of $A_i$

…

$A_2$=`expr $ $A_2$ + 1` ; done )  #loop of $A_2$

$A_1$=`expr $ $A_1$ + 1` ; done  #loop of $A_1$

echo 'universal LUTs are stored in the Hierarchical directories'

############################################################

The following FORTRAN90 code is to read the input variables to run a parameterization. You can easily convert into FORTRAN77. In this example, we assume that all variables are real variables. (You can apply for integer variables as well.). In this example, $A_1, A_2, ..., A_i, A_{i+1}, ..., A_N$ is not digital. You must specify the name of each variable.

############################################################

! Declare types of variables

REAL :: $A_1, A_2, ..., A_i, A_{i+1}, ..., A_N$ ,

& $B_1, B_2, B_3, ..., B_M$

INTEGER :: $A_{i+1}$_INT, …, $A_N$_INT

CHARACTER(2), NAME_i,…,NAME_N

! read the input name

READ(*,*) $A_1, A_2, ..., A_i, A_{i+1}, ..., A_N$

! You can assign proper decimal place, if you need.

$A_i = A_i / @$

! Call for subroutine (This is your parameterization)

CALL PARAM( $A_1, A_2, A_3, ..., A_i, A_{i+1}, ..., A_N$ )

! Convert the input variables into directory name.

$A_{i+1}$ _INT=NINT( $A_{i+1}$ ) WRITE(NAME_i,"(I2.2")  $A_{i+1}$ _INT

…

$A_N$ _INT=NINT( $A_N$ )

WRITE(NAME_N,"(I2.2")  $A_N$ _INT

! Following IF statement is for the case of a variable having

! both single or double digits.

IF ( $A_N$ .LT.10) THEN

N=2

ELSE

N=1

ENDIF

! Generate binary LUTs. M (in RECL) is number of output variables

OPEN(UNIT=1,FILE=NAMEi//…//NAMEN(N:2)//'.dat',

& ACCESS='DIRECT', RECL=M*4)

WRITE(1,REC=1) $B_1, B_2, B_3, ..., B_M$

CLOSE(1)

END

############################################################

**Appendix 2**

The following FORTRAN90 code is to execute the LUT-based approach. You can easily convert into FORTRAN77, too.

############################################################

! Input and output parameters is brought via call-subroutine statement from

! the dynamic core or other parameterization. You can use “common” statement, too.

SUBROUTINE LUT_PARAM ( $A_1, A_2, ..., A_i, A_{i+1}, ..., A_N$ ,

& $B_1, B_2, B_3, ..., B_M$ )

! Value declaration

REAL :: $A_1, A_2, ..., A_i, A_{i+1}, ..., A_N$

& $B_1, B_2, B_3, ..., B_M$

INTEGER :: $A_1$_INT, $A_2$_INT, …,

& $A_i$_INT, $A_{i+1}$_INT, …, $A_N$_INT

! Following character variables are converted from input variables.

CHARACTER(2), NAME_1, NAME_2,…,

& NAME_i, NAME_i1, …, NAME_N

! Convert input variable into directly or file name.

$A_1$_INT =NINT( $A_1$ )

WRITE(NAME_1,"(I2.2)") $A_1$_INT

$A_2$_INT =NINT( $A_2$ )

WRITE(NAME_2,"(I2.2)") $A_2$_INT

…

$A_i$_INT =NINT( $A_i$ )

WRITE(NAME_i,"(I2.2)") $A_i$_INT

$A_{i+1}$_INT =NINT( $A_{i+1}$ )

WRITE(NAME_i1,"(I2.2)") $A_{i+1}$_INT

…

$A_N$_INT =NINT( $A_N$ )

WRITE(NAME_N,"(I2.2)") $A_N$_INT

! Following IF statement is for the case of a variable having both single and double digits.

IF ( $A_N$ .LT.10) THEN

N=2

ELSE

N=1

ENDIF

```
! Specify directory name.

     OPEN(UNIT=1,FILE=NAME_1//NAME_2//…//NAME_i//

! Specify file name.

    &      '/'//NAME_i1//…//NAME_N(N:2)//'.dat',

! Open/read/close a LUT in the assigned directly. M is the number of output variable.

    &      ACCESS='DIRECT', RECL=M*4)

     READ(1,REC=1)  B1, B2, B3,..., BM

     CLOSE(1)

     END

#################################################################
```

**Appendix 3**

**Application for the "data" statement**

1) If your parameterization is small enough to store the entire set of output variables in the FORTRAN code; or 2) if the LUT-approach with open/read/close of binary data file does not significantly replicate the original parameterization (Toshi/Giovanni-this last clause is not clear), you can try to create the multi-dimensional output file in the "data" statement. But in this case you have to convert the input file into the integer variables that start from 1. The computational time of this method becomes even cheaper, since the input (open/read/close) statement is omitted from the FORTRAN code. If the computational memory permit us to apply this method for the NFL code, the estimated wall-clock time of this process is only 0.05 (seconds) for 5000 repetitive process, and it is approximately *3000 times* faster than the original NFL code.. However, the compilation time of the code will be significantly longer than the aforementioned binary-LUT method.

############################################################

! Input and output parameters is brought via call-subroutine statement from

! the dynamic core or other parameterization. You can use "common" statement, too.

SUBROUTINE LUT_PARAM ( $A_1, A_2, ..., A_i, A_{i+1}, ..., A_N$ ,

& $B_1, B_2, B_3, ..., B_M$ )

! Value declaration

REAL :: $A_1, A_2, ..., A_i, A_{i+1}, ..., A_N$ ,

& $B_1, B_2, B_3, ..., B_M$

& OUTPUT(M)

INTEGER :: $A_1$_INT, $A_2$_INT, …,

& $A_i$_INT, $A_{i+1}$_INT, …, $A_N$_INT

! Here you store the universal output in the memory when compiled.

DATA ((…(OUT ( $A_1, A_2, ..., A_i, A_{i+1}, ..., A_N$ , M) ,

& $A_1$ = 1, @), $A_2$ = 1, @)…, $A_N$ =1,@)

& / $B_1, B_2, B_3, ..., B_M$ ,

….

& $B_1, B_2, B_3, ..., B_M$ /

! Convert input variable (real) into dimension variable (integer).

$A_1$_INT =NINT( $A_1$ )

$A_2$_INT =NINT( $A_2$ )

…

$A_i$_INT =NINT( $A_i$ )

$A_{i+1}$_INT =NINT( $A_{i+1}$ )

…

$A_N$ _INT =NINT( $A_N$ )

! Derive the output variable from the DATA statement.

$B_1$ = OUT ( $A_1, A_2, ..., A_i, A_{i+1}, ..., A_N$ , 1)

$B_2$ = OUT ( $A_1, A_2, ..., A_i, A_{i+1}, ..., A_N$ , 2)

…

$B_M$ = OUT ( $A_1, A_2, ..., A_i, A_{i+1}, ..., A_N$ , M)

END

##########################################################

**References**

Fu, Q., and K.N. Liou, On the correlated k-distribution method for radiative transfer in nonhomogeneous atmospheres. *J. Atmos. Sci.*, *49*, 2139-2156, 1992.

Fu, Q., and K.N. Liou, Parameterization of the radiative properties of cirrus clouds. *J. Atmos. Sci.*, *50*, 2008-2025, 1993.

Mahrer, Y., and R.A. Pielke, A numerical study of the airflow over irregular terrain. *Beitrage zur Physik der Atmosphere, 50*, 98-113, 1977.

Pielke, R.A. Sr.., Mesoscale meteorological modeling. 1st Edition, Academic Press, New York, N.Y., 612 pp., 1984.

Pielke, R.A, Sr.., Mesoscale meteorological modeling. 2nd Edition, Academic Press, San Diego, CA, 676 pp., 2002.

Pielke, R.A., W.R. Cotton, R.L. Walko, C.J. Tremback, W.A. Lyons, L.D. Grasso, M.E. Nicholls, M.D. Moran, D.A. Wesley, T.J. Lee, and J.H. Copeland, A comprehensive meteorological modeling system -- RAMS. *Meteor. Atmos. Phys., 49*, 69-91, 1992.

Rose, F.G. and Charlock, T.P., *New Fu-Liou code tested with ARM raman lidar aerosols and CERES in pre-CALIPSO sensitivity study.* 11th Conf. on Atmos Radiation, Ogden, Orville 1980 see B-1, 2002.

Rosen H. K., R. R. Rosinski, J. M. Farber., and D. A. Host., UNIX System V Release 4: An Introduction, 2nd Edition, Osborn McGraw-Hill, Berkeley, CA, 1175pp., 1996.

Stephens, G.. L., Parameterization of radiation for numerical weather prediction models. *Mon. Wea. Rev.* 112, 826-867, 1984.

Walko, R.L., W.R. Cotton, M.P. Meyers, and J.Y. Harrington, New RAMS cloud microphysics parameterization. Part I: the single-moment scheme. *Atmos. Res.*, *38*, 29-62, 1995.

Wang J., U. S. Nair and S. A. Christopher, GOES-8 Aerosol Optical Thickness Assimilation in a Mescoscale Model, *J. Geophys. Res.,* 2004 (Submitted).